CMSC 733 - Project 3
Buildings built in minutes: An SfM Approach Using 2 late days

Nishad Kulkarni
A. James Clark, School of Engineering
University of Maryland, College Park
UID - 117555431, Email: nkulkar2@umd.edu

Saurabh Palande
A. James Clark, School of Engineering
University of Maryland, College Park
UID - 118133959, Email: spalande@umd.edu

I. Introduction

In this project we try to reconstruct a 3D scene and simultaneously obtain the camera poses. An imageset of a building using a monocular camera is used.

II. Get Inliers

The matches between every pair of images from 1 to 6 is provided in a text file. These matches are extracted and the RANSAC algorithm is applied to reject outliers. Since the error of these matches themselves is very low and of the order of 10e-30, the threshold itself was set to 0 and matches that have zero SSD error, were utilized. This algorithm is the same as applied in the first project - 'MyAutoPano'.

The comprehensive code for this task is titled `'GetInliersRANSAC.py'`. The feature correspondence for images is shown in the figures below:

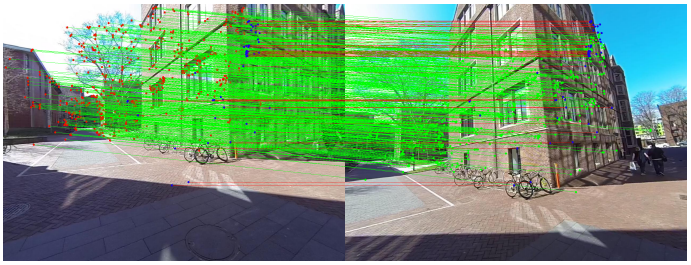

Fig. 1. Feature correspondence between image 1 and 2

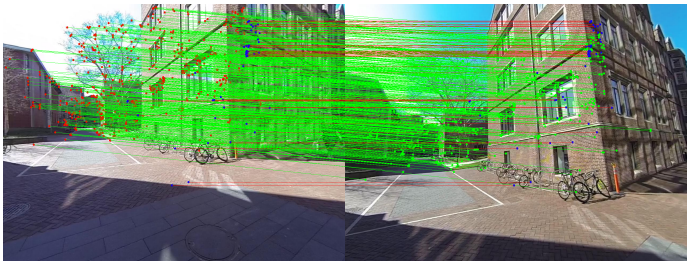

Fig. 2. Feature correspondence between image 1 and 3

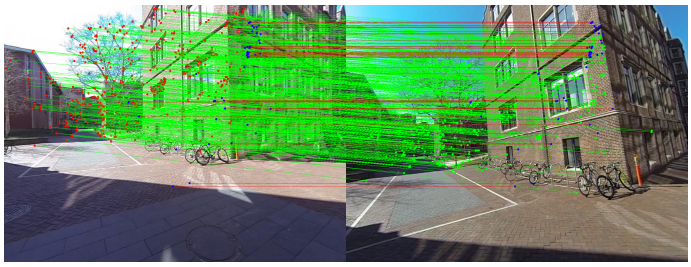

Fig. 3. Feature correspondence between image 1 and 4

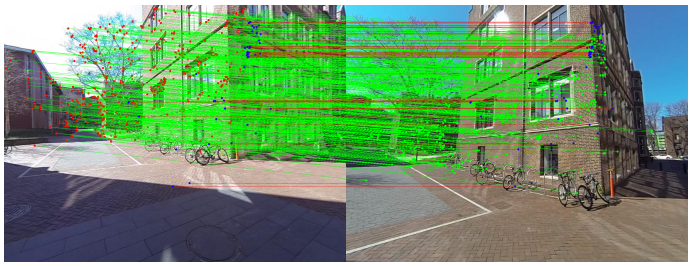

Fig. 4. Feature correspondence between image 1 and 5

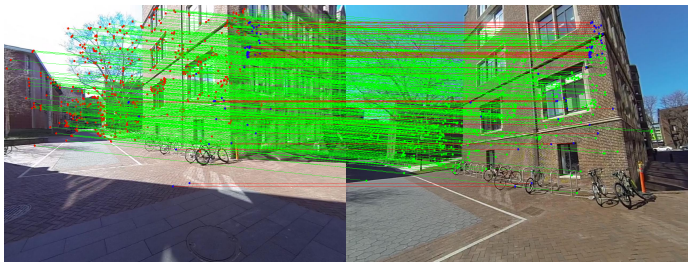

Fig. 5. Feature correspondence between image 1 and 6

III. Estimate Fundamental Matrix and Essential Matrix

A. Fundamental Matrix

Using the inliers fouind in the previous section, we now estimate the Fundamental Matrix. The epipolar constraint is used to calculate the values in the Fundamanetal Matrix. Since there may be noise in the Fundamental Matrix, it's rank may

be 3, we then force the Matrix to be of rank 2. This is acheived by following these steps:

1) Use all the matches in the Epipolar constraint to get the Fundamental matrix for given pair.
2) Get the U,S,V components of the Singular Value Decompositon of 'F'.
3) Set the last value in the diagonal matrix 'S' as 0.
4) Multiply U, new S, and V to get new Fundamental matrix of rank 2.

The comprehensive code for this task is titled `'EstimateFundamentalMatrix.py'`. The fundamental matrix is shown in the figure below:

```
********** Fundamental Matrix **********
array([[-3.68883302e-07, -1.10477846e-05,  2.86970615e-03],
       [ 1.40698432e-05, -1.24627689e-06, -3.23532667e-03],
       [-4.88006457e-03,  9.18311221e-04,  1.00000000e+00]])
```

Fig. 6. Fundamental Matrix

B. Essential Matrix

Now that we have the Fundamental Matrix, and given the Calibration Matrix 'K', we can calculate the Essential Matrix. There is again a chance that the Essential Matrix might have some noise and thus be of rank 3, to avoid this the same procedure as above is used to force the rank to be 2.

The comprehensive code for this task is titled `'EssentialMatrixFromFundamentalMatrix.py'`. The essential matrix is shown in the figure below:

```
********** Essential Matrix **********
array([[-0.0196783 , -0.64312647, -0.26970459],
       [ 0.82575626, -0.07723867,  0.53667266],
       [ 0.16791851, -0.69507518, -0.16148575]])
```

Fig. 7. Essential Matrix

IV. Camera Pose Estimation

The camera pose is decompose to get 4 camera configurations: (C_1,R_1),(C_2,R_2),(C_3,R_3), and (C_4,R_4). Only one of these is the correct pose, which can be found by traingulation.

The comprehensive code for this task is titled `'ExtractCameraPose.py'`.

V. Triangulation

A. Linear Triangulation

We find the 3D points using linear least squares. This is impelemented in the file `'LinearTriangulation.py'`.

Of the 4 cametra poses found in the previous section, one pair gives a point-cloud with negative depth while the other gives negative depth. We use Linear Least Squares to check for the cheirality condition.

This is acheived in the code titled `'DisambiguateCameraPose.py'` Linear traingulation for 4 camera poses is shown in the below figure:

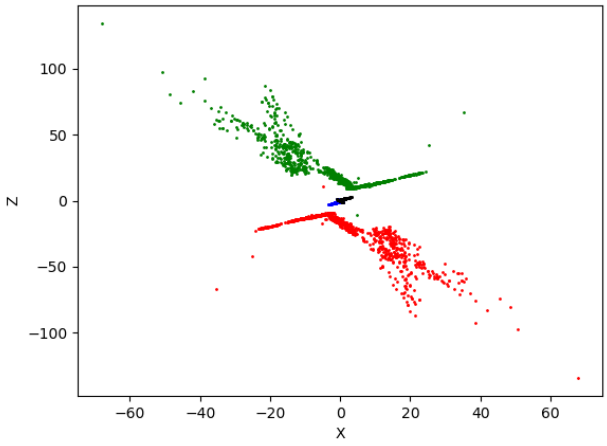

Fig. 8. Linear triangulation

VI. Camera Pose Estimation

A. Non-Linear Triangulation

The point cloud found by linear triangulation can be further refined by narrowing it down to a set that minimizes the reprojection error instead of algebraic error.

This is acheived in the code titled `'NonlinearTriangulation.py'`. The output of Non - Linear traingulation is shown in the below figure:

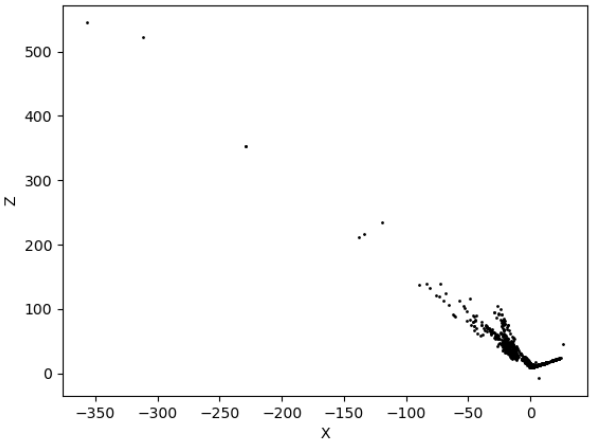

Fig. 9. Non - Linear Triangulation

VII. Perspective-n-Points

A. Linear Camera Pose Estimation

Here we try to isolate the camera parameters (*C*,*R*). The rotation matrix *R* need to be forced to be orthogonal. This can be acheived by finding the SVD of R as $R = UDV^T$ and then getting the corrected $R = UV^T$. Also, $R = sign(|R|) * R$.

B. PnP Ransac

Here, we simply implement RANSAC again to reject outliers. This time however, we minimise reprojection errors.

C. Nonlinear PnP

Most of the errors are Non linear and must thus be minimised accordingly. For this least squares is implemented.

VIII. Bundle Adjustment

We finally try to refine the output.

A. Visibility Matrix

Here, the visibility of a point from a given camera pose is checked.

B. Bundle Adjustment

Finally, we try to minimise reprojection error over by checking the visibility of a point from a given camera pose using it's extrinsic parameters.

The output of bundle adjustment is shown in the below figures:

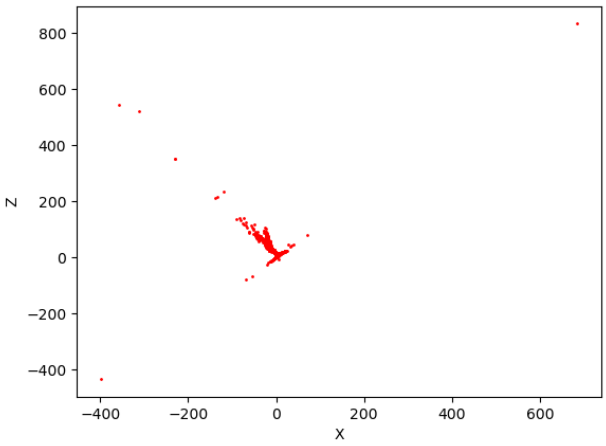

Fig. 10. Bundle Adjustment

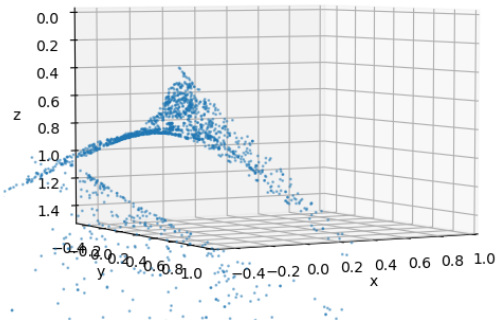

Fig. 11. Bundle Adjustment 3D